Research Journal of Applied Sciences, Engineering and Technology 11(12): 1417-1423, 2015
DOI: 10.19026/rjaset.11.2249
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: July 24, 2015 Accepted: August 30, 2015 Published: December 25, 2015

Research Article

Performance Analysis of an Embarrassingly Parallel Application in Atmospheric Modeling

M. Varalakshmi and Daphne Lopez
VIT University, Vellore, India

Abstract: This study aims at making a comparative study of various parallel programming models for a compute intensive application pertaining to Atmospheric modeling. Atmospheric modeling deals with predicting the behavior of atmosphere through mathematical equations governing the atmospheric fluid flows. The mathematical equations are nonlinear partial differential equations which are difficult to solve analytically. Thus fundamental governing equations of atmospheric motion are discretized into algebraic forms that are solved using numerical methods to obtain flow-field values at discrete points in time and/or space. Solving these equations often requires huge computational resource, which is normally available with high-speed supercomputers. Shallow Water equations provide a useful framework for the analysis of dynamics of large-scale atmospheric flow and for the analysis of various numerical methods that might be applied to the solution of these equations. In this study, Finite volume approach has been used for discretizing these equations that leads to a number of algebraic equations equal to the number of time instants at which the flow field values are to be evaluated. It is apparent that the application is embarrassingly parallel and its parallelization will suppress communication overhead. A High Performance Compute cluster has been employed for solving the equations involved in atmospheric modeling. Use of OpenMP and MPI APIs has paved the way to study the behavior of shared memory programming model and the message passing programming model in the context of such a highly compute intensive application. It is observed that no additional benefit can be enjoyed by creating too many software threads than the available hardware threads, as the execution resources should be shared among them.

Keywords: Atmospheric modeling, MPI, OpenMP, parallel computing, shallow water equations

INTRODUCTION

Parallel Computing find its application in various disciplines. In Science and Engineering, parallel computing has been considered to be the high end of computing and has been used to model difficult problems in many areas such as Atmosphere, Earth, Environment Physics nuclear, particle, condensed matter, high pressure, fusion, photonics Bioscience, Biotechnology, Genetics Chemistry, Molecular Sciences, Geology, Seismology, Mechanical Engineering-from prosthetics to spacecraft, Electrical Engineering, Circuit Design, Microelectronics, Computer Science and Mathematics. In Industrial and Commercial discipline, commercial applications provide an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways (Li *et al*., 2005). Such applications are Databases, data mining, Oil exploration, web search engines, web based business services, Medical imaging and diagnosis, Pharmaceutical design, Financial and economic modeling, Management of national and multi-national corporations, Advanced graphics and virtual reality, particularly in the entertainment industry, Networked video and multi-media technologies, Collaborative work environments. In the field of Atmospheric and Ocean Simulation, Climate Modeling and Weather Prediction also involves a lot of compute intensive and number crunching operations, which can be solved by way of parallel processing with great speed and high accuracy (Marowka, 2008).

Shared memory model: In this model, there is one (large) common shared memory for all processors. The key feature is the use of a single address space across the whole memory system, so that all the processors have the same view of memory. The processors communicate with one another by one processor writing data into a location in memory and another processor reading the data. With this type of communications the time to access any piece of data is the same, as all of the communication goes through the bus. The advantage of this type of architecture is that it is easy to program as there are no explicit communications between processors, with the communications being handled via the global memory store. Access to this memory store can be controlled using techniques developed from multi-tasking computers, e.g., semaphores. However, the shared

Corresponding Author: Daphne Lopez, VIT University, Vellore, India

memory architecture does not scale well. The main problem occurs when a number of processors attempt to access the global memory store at the same time, leading to a bottleneck. One method of avoiding this is memory access conflict is by dividing the memory into multiple memory modules, each connected to the processors via a high performance switching network. However, this approach tends to shift the bottleneck to the communications network (Bethune *et al.*, 2013). OpenMP-Open Multi-Processing (OpenMP) is the API that supports shared memory (Quinn, 2003). In relation to other parallel programming techniques it lies between HPF (High Performance Fortran) and MPI in that it has the ease of use of HPF, in the form of compiler directives, combined with the functionality of MPI.

Message passing model: In this model, each processor has its own (small) local memory and its content is not replicated anywhere else. Parallel tasks exchange data through passing messages to one another. These communications can be asynchronous or synchronous. MPI is a message-passing interface (Quinn, 2003), together with protocol and semantic specifications. It is supported on virtually all HPC platforms such as a cluster (Sterling, 2001) or a network of workstations.

MPI-OpenMP model: The nodes of the Clusters today are at least dual-processor Symmetric Processing (SMP) systems. So, each node may have more than one core within it. Parallel programming may combine the distributed memory parallelization on the node interconnect with the shared memory parallelization inside of each node. Hybrid MPI and OpenMP (Shan, 2011) approach is deployed to work with such clusters whereby OpenMP is used for data sharing among the multi-cores that comprise a node and MPI is used for communication between nodes (Artés *et al.*, 2013). These various models can be deployed for solving highly compute intensive applications especially the equations describing the atmospheric fluid flow. Thus to begin with, simpler two-dimensional system of governing equations of atmospheric fluid flow, termed as shallow water equations have been implemented in these various models.

Shallow water equations: Any fluid flow (including air) is characterized by the properties such as velocity, density, pressure, temperature and its space and time derivatives. Equations describing any fluid flow characteristics are referred to as governing equations of motion. Many of the mathematical and computational properties of these governing equations can be embodied in a simpler two-dimensional system of equations known as shallow water equations (Jacobson, 2005). The shallow water equations consider the fluid to be rotating, homogeneous, incompressible and hydrostatic with a finite free surface height. They are derived by assuming that the horizontal length scale is much greater than the vertical length scale. (i.e., by approximating the atmosphere to a fluid of limited depth). This assumption implies that the density as well as the horizontal velocity field is constant throughout the depth of the fluid (Hack and Jakob, 1992).

Shallow Water equations provide a useful framework for the analysis of the dynamics of large-scale atmospheric flow and for the analysis of various numerical methods that might be applied to the solution of these equations. Moreover, Situations in fluid dynamics where the horizontal length scale is much greater than the vertical length scale are common, so the shallow water equations are widely applicable. Particularly, the horizontal momentum (dV/dt) and mass continuity (dϕ/dt) equations of the shallow water system have been considered for implementation in this project. The horizontal velocity field can be represented in terms of the vertical component of the relative velocity, ζ and horizontal divergence, δ.

Vorticity: It refers to the vertical component of the curl of the wind and is a measure of the "spin" of the wind about a vertical axis, with counter clockwise spin being positive. Including the effect of the Earth's rotation to the relative vorticity, gives the absolute vorticity, η (Hack and Jakob, 1992; Jacobson, 2005). Thus the absolute vorticity has two contributing terms: the vorticity associated with the wind and the vorticity associated with the spin of the Earth. They are called the relative vorticity and the planetary vorticity respectively. The planetary vorticity is exactly equal to the Coriolis parameter f. Hence, the absolute vorticity $\eta = \zeta + f$.

Horizontal divergence: It is the fractional rate of increase of an element of area of a marked fluid particle (Jacobson, 2005).

Geopotential: Free surface Geopotential, ϕ is the potential of the Earth's gravity field. Φ = gh, where ϕ is the geopotential at height, h and g is the gravitational constant. Therefore, the following equations related to Vorticity, Horizontal Divergence, rate of change of Vorticity, rate of change of Divergence, rate of change of Geo-potential Height will be considered for further implementation (Jacobson, 2005):

$$\eta = \frac{1}{R_E(1-\mu^2)}\frac{\partial V}{\partial \lambda} - \frac{1}{R_E}\frac{\partial U}{\partial \mu} + f \tag{1}$$

$$\delta = \frac{1}{R_E(1-\mu^2)}\frac{\partial U}{\partial \lambda} + \frac{1}{R_E}\frac{\partial V}{\partial \mu} \tag{2}$$

$$\frac{\partial \eta}{\partial t} = -\frac{1}{R_E(1-\mu^2)}\frac{\partial (U\eta)}{\partial \lambda} - \frac{1}{R_E}\frac{\partial (V\eta)}{\partial \mu} \tag{3}$$

$$\frac{\partial \delta}{\partial t} = \frac{1}{R_E(1-\mu^2)}\frac{\partial B}{\partial \lambda} - \frac{1}{R_E}\frac{\partial A}{\partial \mu} - \nabla^2\left(\Phi + \frac{U^2+V^2}{2(1-\mu^2)}\right) \tag{4}$$

$$\frac{\partial \Phi}{\partial t} = -\frac{1}{R_E(1-\mu^2)}\frac{\partial (U\Phi)}{\partial \lambda} - \frac{1}{R_E}\frac{\partial (V\Phi)}{\partial \mu} - \Phi\delta \qquad (5)$$

The first two equations are termed diagnostic equations as they do not involve any time derivatives. The last three equations are termed prognostic equations as they include time derivatives and hence used for forecasting.

METHODOLOGY

Discretization: Finite Volume method is as efficient as the finite element method for its application in irregular grids too. However, it is as simple as the Finite Difference method in formulating the equations.
Other advantages of the Finite volume method are:

- Do not require any coordinate transformations for irregular shapes unlike the finite difference method.
- Treats arbitrary geometries efficiently. Hence, well suited for two and three dimensional flow computation
- Use of Integral formulation provides more natural treatment of boundary conditions
- Naturally applied to PDEs for expressing conservation laws.

Hence FDM can be considered to be superior to other elementary discretization methods such as the Finite Difference Method, FDM and the Finite Element Method, FEM.

Finite volume discretization: Discretization refers to replacing the partial derivatives in the governing equations of motion with the algebraic terms. The equations of vorticity, divergence and geopotential mentioned above, are nonlinear partial differential equations. Analytical solutions of PDEs provide the variation of the dependent variables continuously throughout the domain. But solving the PDEs over a huge domain such as the atmospheric domain analytically is very tedious. Thus these partial differential equations are discretized (converted) into algebraic forms that are solved using numerical methods. However, the numerical solutions provide the flow-field values only at discrete points in time and/or space called grid points.

In FVD, first the entire fluid flow domain is divided into several discrete control volumes. The nodal points or the grid points, at which the flow properties are to be evaluated, are assumed to be at the centres of these control volumes.

Figure 1, the points marked as N, S, E, W, P are the nodal points or the grid points which are assumed to be equally spaced and surrounded by a discrete control volume (Fig. 2 and 3).

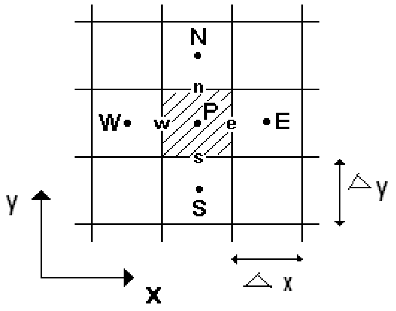

Fig. 1: Structured finite volume grid

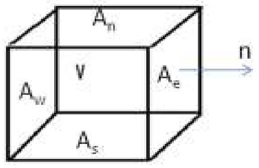

Fig. 2: Finite volume grid cell in 3D

Fig. 3: Finite volume grid cell in 2D

Next, the governing equations of motion are integrated over each sub domain. Gauss Divergence theorem is used for the integration.

Gauss divergence or green's theorem: It states that the integral of a derivative over a region is equal to the value of the function at the boundary of the region. It means the rate of change of a flow property within the control volume is equivalent to the flux crossing the surface S of volume V. i.e., the net flux flow out of a region is equal to sum of all sources-sum of all sinks. This theorem is applied over a control volume to get the divergence of a vector field. In 3D-Version of Divergence Theorem, if V is the volume bounded by a closed surface S and A is a vector function of position with continuous derivatives then.

$$\iiint_V \nabla.\mathrm{A}dV = \frac{1}{\Delta V}\iint_S \mathrm{A.n\,ds} = \frac{1}{\Delta V}\iint \mathrm{A.ds} = \frac{1}{\Delta V}\sum_{i=1}^{N} A_i a_i$$

where,

n	= Outward drawn normal to S
∇A	= Flux or net outflow per unit volume of the vector A through the surface ΔS
N	= Number of surfaces bounding the 3-dimensional control volume
ΔV	= Volume of a single control volume
a_i	= Area of the surface 'i'

In 2D-Version of Divergence Theorem, the surface integral changes into a line integral:

$$\underset{S}{\iiint} \nabla A \, dA = \frac{1}{\Delta A} \underset{C}{\int} A.n \, ds = \frac{1}{\Delta A} \sum_{i=1}^{N} A_i l_i$$

where,

l_i = Length of the side 'i'
N = Number of sides bounding the 2-dimensional control volume

OpenMP Application program interface: The OpenMP-Open Multi-Processing is an API that supports Multi-Platform (UNIX, Windows) and Shared Memory programming. Programmer need not specify the processors (nodes) on which to execute the task. OpenMP Programming can be performed in C, C++ and FORTRAN. It is a portable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer. It is composed of a set of compiler directives, library routines and environment variables. But scalability is hindered due to shared memory architecture.

The standard view of parallelism in a shared-memory program is fork-join parallelism. When the program starts execution, only a single thread called the master thread is active. It executes the sequential portions of the algorithm. At those points where parallel operations are required, the master thread forks additional threads. The master thread and the created threads work concurrently through the parallel section. At the end of the parallel code the created threads die or are suspended and the flow of control returns to the single master thread. This is called a join (Sato, 2002; Zheng *et al*., 2011). Thus in any shared memory model, the number of active threads is one at the program's start and finish and may change dynamically throughout the execution of the program. This model supports incremental parallelization where in the sequential program is transformed into a parallel program one block of code at a time (Amit *et al*., 2012).

MPI Application program interface: Each processor in the Message Passing model has direct access only to the instructions and data stored in its local memory but can exchange messages among themselves via the interconnection network (Diaz *et al*., 2012). The user specifies the number of concurrent processes when the program begins and typically the number of active processes remains constant throughout the execution of the program. MPI-Message Passing Interface is an Application Program Interface together with protocol and semantic specifications. Message passing libraries allow efficient parallel programs to be written for distributed memory systems. These libraries provide routines to initiate and configure the messaging environment as well as sending and receiving packets of data. Currently, there are several implementations of MPI (Dekate *et al*., 2012), including versions for networks of workstations, clusters of personal computers, distributed-memory multiprocessors and shared-memory machines. MPI enhances performance in clusters (Sterling, 2001).

It exhibits a high level of scalability and portability. The goal of portability, architecture and network transparency has been achieved with the low-level communication library like MPI. The library provides an interface for C and FORTRAN and additional support of graphical tools. However, these message-passing systems are still stigmatized as low-level because most tasks of the parallelization are still left to the application programmer. When writing parallel applications using message passing, the programmer still has to develop a significant amount of software to manage some of the tasks of the parallelization, such as: the communication and synchronization between processes, data partitioning and distribution, mapping of processes onto processors and input/output of data structures. If the application programmer has no special support for these tasks, it then becomes difficult to widely exploit parallel computing. The easy-to-use goal is not accomplished with a bare message-passing system and hence requires additional support. It also lays the burden of assigning the nodes on the programmer.

In MPI, processes belong to groups. If a group contains n processes, then each of the processes in the group is identified within the group by a rank (id), which is an integer from 0 to n-1 (Nupairoj and Ni, 1994). A process may belong to more than one group. There is an initial group to which all processes in an MPI implementation belong. Such a group forms a so-called communication domain. A communication domain is a set of processes that are allowed to communicate with each other. Each process can belong to many different (possibly overlapping) communication domains. The communication domain also called a communicator is used to define a set of processes that can communicate with each other. Processes executing in parallel have different address spaces and during communication between processes a part of the data in the address space of the sending process is copied in the address space of the receiving process (Hursey *et al*., 2007). Therefore, communication is achieved by sending and receiving messages.

The aforementioned models can be deployed for solving highly compute intensive applications such as the equations describing the atmospheric fluid flow. In this study, simpler two-dimensional system of governing equations of atmospheric fluid flow, termed as shallow water equations have been considered for implementation in these various models.

Table 1: Execution time for varying number of cores in OpenMP, MPI and Hybrid MPI-OpenMP

Parallel models		No. of execution cores							
		1	2	4	6	8	10	12	16
Execution time (secs)	OpenMP	422	191	141	124	115	118	122	136
	MPI	432	1380	480	394.8	372.6	201.6	186	153.6
	Hybrid MPI- OpenMP	366		301.2		417	438.6		

IMPLEMENTATION

MERRA: Modern-Era Retrospective Analysis for Research and Applications is a NASA reanalysis for the satellite era using a major new version of the Goddard Earth Observing System Data Assimilation System Version 5 (GEOS-5). The Project focuses on historical analyses of the hydrological cycle on a broad range of weather and climate time scales and places the NASA EOS suite of observations in a climate context. MERRA data is available at the Modeling and Assimilation Data and Information Services Center (MDISC) managed by the NASA Goddard Earth Sciences (GES) Data and Information Services Center (DISC). The data for the two-dimensional velocity components, u and v and the geo-potential height, h have been obtained from MERRA data. This data has been provided for 361 nodal points along the latitude and 540 nodal points along the longitude. Thus this project work tries to solve the shallow water equations for a total of 540×361 grid points to suit the input MERRA data for u, v and h.

Solution domain for consideration: In this study, the flow domain considered for solving the equations is-180° to 180° E Longitude and -90° to 90° N Latitude. The total 180° Latitude is partitioned into 361 subdivisions and the total 360° Longitude is partitioned into 540 subdivisions to map with the input data obtained from MERRA. The vorticity, divergence and the mass continuity values are evaluated and forecasted for every 30 sec for a total of 3 days over a flow domain grid of size 361×540. The 361×540 size 2-D arrays needed for storing the vorticity, divergence and the free surface geo-potential values are declared inside a structure. Due to memory constraints, distinct arrays are not created for every 30 sec time instant. Instead only two structure variables are created.

If the vorticity values for the current time instant are evaluated and stored in the structure variable t[0], then the values for the next time instant are evaluated using this previous time instant data and stored in the other structure variable, t[1]. Also these values are stored in to a file as and when computed. Thus the computation for the third time instant is done and the results are stored in t[0] by overwriting the previous contents. This is continued for all the 72×120 time instants. The file writing is done for every array computed instead of for every single nodal point to minimize the number of file I/O operations. Moreover a binary file would occupy less space than a text file. It is to be noted that for computing the flow field value at any particular grid point, the field values at the neighboring cells in all the four directions should be known. In such a scenario, to compute the values at the end points say 0 and 361 in case of latitude and 0 and 540 in case of longitude, there are no adjacent cells for one direction as such. Hence, while simulating the calculation, the corner points can be neglected and calculation for the remaining points can be done.

RESULTS AND DISCUSSION

The results of evaluating the shallow water equations have been plotted using MATLAB which can be used for easy interpretation of the atmospheric flow field values. Also, Table 1 illustrates the execution time required for varying number of threads in all three models of programming. Figure 4 shows the plot of execution time needed for 72×120 forecasts (for a period of 3 days) with varying number of threads in shared memory model. Likewise graphs plotted for pure MPI and hybrid programming models are shown in Fig. 5 and 6. The number of cores has been taken along the x-axis and the execution time along the y-axis.

When the code is executed with just two threads, it brings about a speed up of 45% and the execution time reduces gradually upon increasing the number of threads up to 8. However, in case of over threading where in the number of threads range from 10 to 16, it becomes slower than when executed with lesser number of threads. This may be owing to the fact that embarrassingly parallel applications in general keep the execution resources busy and hence no additional benefit can be enjoyed by sharing the execution resources among the software threads that are more in number than the hardware threads. Nevertheless, execution is faster than the sequential code execution.

In case of MPI, as message passing is involved, the communication overheads involved, makes its execution time quite higher than that of OpenMP implementation. But the time is found to be nearly equal with 16 threads, in both the cases. With 2 threads, the execution time is even worse than sequential, accounting for an increase of processing time by more than 3 folds. This is because of the master-slave approach followed in the implementation. Hence with 2 threads, one acts as the master and only the other one works on the data. It only incurs an additional

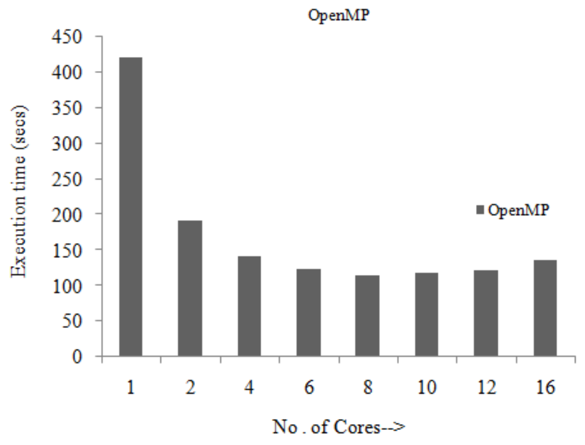

Fig. 4: Plots for execution time in OpenMP

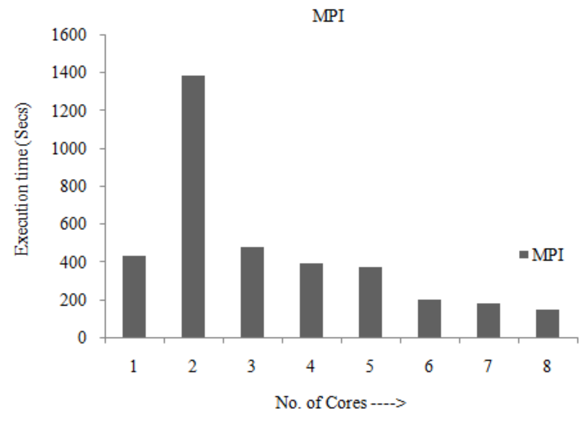

Fig. 5: Plots for execution time in MPI

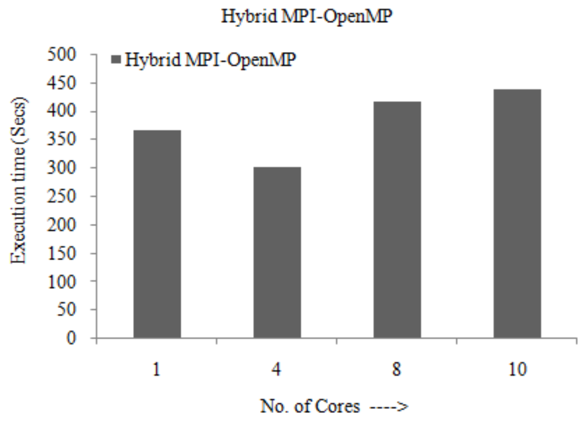

Fig. 6: Plots for execution time in Hybrid MPI-OpenMP

communication overhead resulting in the worst performance. Hybrid MPI-OpenMP programming model has offered a speedup of 40%.

CONCLUSION

In OpenMP implementation, creating more software threads than the available hardware threads does not bring about any additional efficiency. In contrary it slows down the execution. This may be due to the reason that the application is compute-intensive rather than communication-intensive. In case of MPI, although, the execution time is quite higher than that of OpenMP implementation, MPI offers more scalability compared to OpenMP as OpenMP can be used only for multiple cores within a single system which is difficult to extend beyond 16 cores. Hybrid MPI-OpenMP programming model has been proven to work well with clusters and a network of workstations. As Shallow water equation models have only one vertical level, complex Navier-Stoke's equations can be solved not only to make the results useful for actual weather prediction but also to fully experience the benefit of using a cluster.

REFERENCES

Amit, A., T. Danesh, L. Rui, K. Rick and C. Barbara, 2012. OpenMP parallelism for fluid and fluid-particulate systems. Parallel Comput., 38(9): 501-517.

Artés, T., A. Cencerrado, A. Cortés and T. Margalef, 2013. Relieving the effects of uncertainty in forest fire spread prediction by hybrid MPI-OpenMP parallel strategies. Proc. Comput. Sci., 18: 2278-2287.

Bethune, I., J.M. Bull, N.J. Dingle and N.J. Higham, 2013. Performance analysis of asynchronous Jacobi's method implemented in MPI, SHMEM and OpenMP. Int. J. High Perform. C., 28(1): 97-111.

Dekate, C., M. Anderson, M. Brodowicz, H. Kaiser, B. Adelstein-Lelbach and T. Sterling, 2012. Improving the scalability of parallel *N*-body applications with an event-driven constraint-based execution model. Int. J. High Perform. C., 26(3): 319-332.

Diaz, J., C. Munoz-Caro and A. Nino, 2012. A survey of parallel programming models and tools in the multi and many-core era. IEEE T. Parall. Distr., 23(8): 1369-1386

Hack, J.J. and R. Jakob, 1992. Description of a global shallow water model based on the spectral transform method. NCAR Technical Note, NCAR/TN-343+STR, pp: 93.

Hursey, J., J.M. Squyres, T.I. Mattox and A. Lumsdaine, 2007. The design and implementation of checkpoint/restart process fault tolerance for open MPI. Proceeding of the IEEE International Parallel and Distributed Processing Symposium (IPDPS, 2007), pp: 1-8

Jacobson, M.Z., 2005. Fundamentals of Atmospheric Modeling. 2nd Edn., Cambridge University Press, New York.

Li, J., J. Shu, C. Yongjian, W. Dingxing and W. Zheng, 2005. Analysis of factors affecting execution performance of OpenMP programs. Tsinghua Sci. Technol., 10(3): 304-308.

Marowka, A, 2008. Think parallel: Teaching parallel programming today. IEEE Distrib. Syst. Online, 9(8): 1-1.

Nupairoj, N. and L.M. Ni, 1994. Performance evaluation of some MPI implementations on workstation clusters. Proceeding of the Scalable Parallel Libraries Conference, pp: 98-105.

Quinn, M.J., 2003. Parallel Programming in C with MPI and OpenMP. 1st Edn., McGraw-Hill, New York.

Sato, M., 2002. OpenMP: Parallel programming API for shared memory multiprocessors and on-chip multiprocessors. Proceeding of the 15th International Symposium on System Synthesis, pp: 109-111.

Shan, H., 2011. Hybrid Programming for multicore processors, Computational Sciences and Optimization (CSO). Proceeding of 4th International Joint Conference on Computational Sciences and Optimization, pp: 261-262.

Sterling, T., 2001. Beowulf Cluster Computing with Linux. 1st Edn., MIT Press, Cambridge, MA.

Zheng, Z., X. Chen, Z. Wang, L. Shen and J. Li, 2011. Performance model for OpenMP parallelized loops. Proceeding of 2011 International Conference on Transportation, Mechanical and Electrical Engineering (TMEE), pp: 383-387.